Computing
https://doi.org/10.1007/s00607-018-0663-0

Cluster-based vulnerability assessment of operating systems and web browsers

Yazdan Movahedi[1] · **Michel Cukier**[1] · **Ambrose Andongabo**[2] · **Ilir Gashi**[2]

Received: 25 September 2017 / Accepted: 15 September 2018

Abstract
Organizations face the issue of how to best allocate their security resources. Thus, they need an accurate method for assessing how many new vulnerabilities will be reported for the operating systems (OSs) and web browsers they use in a given time period. Our approach consists of clustering vulnerabilities by leveraging the text information within vulnerability records, and then simulating the mean value function of vulnerabilities by relaxing the monotonic intensity function assumption, which is prevalent among the studies that use software reliability models (SRMs) and nonhomogeneous Poisson process in modeling. We applied our approach to the vulnerabilities of four OSs (Windows, Mac, IOS, and Linux) and four web browsers (Internet Explorer, Safari, Firefox, and Chrome). Out of the total eight OSs and web browsers we analyzed using a power-law model issued from a family of SRMs, the model was statistically adequate for modeling in six cases. For these cases, in terms of estimation and forecasting capability, our results, compared to a power-law model without clustering, are more accurate in all cases but one.

Keywords Vulnerability assessment · Nonhomogeneous Poisson process · Clustering · Software reliability models · Software reliability growth · Security growth models

Mathematics Subject Classification 62H30 · 68M15

✉ Ilir Gashi
ilir.gashi.1@city.ac.uk

Yazdan Movahedi
ymovahed@umd.edu

Michel Cukier
mcukier@umd.edu

Ambrose Andongabo
ambrose.andongabo.1@city.ac.uk

[1] University of Maryland, College Park, USA

[2] City, University of London, London, UK

Published online: 25 September 2018

1 Introduction

Security decision makers often use public data sources to help make better decisions on, for example, selecting security products, checking for security trends, and estimating when new vulnerabilities that affect their installations will be publicly reported. Several studies have applied software reliability models (SRMs) to estimate times between public reports of vulnerabilities [1–6]. The studies we are aware of estimate all vulnerabilities together. We postulate that such analysis may miss some insights that apply to separate categories of vulnerabilities, rather than all vulnerabilities together. Moreover, SRMs assume vulnerability detection to be an independent process. However, this might not be the case due, for example, to the discovery of a new type of vulnerability that might prompt attackers to look for similar vulnerabilities [7]. This may lead to less accurate predictions on the next reporting date of vulnerability, or the total number of new vulnerabilities reported in the next time interval. One way to mitigate these issues is to split vulnerabilities into separate clusters and test whether the clusters are independent.

In this paper we present an approach that does the following:

- Uses existing clustering techniques to group vulnerabilities into distinct clusters, using the textual information reported in the vulnerabilities as a basis for constructing the clusters;
- Uses existing SRMs to make predictions on the number of new vulnerabilities that will be discovered in a given time period for each cluster for a given OS/web browser;
- Superposes the SRMs used for each cluster together into a single model for predicting the number of vulnerabilities that will be discovered in a given time period for a given OS/web browser.

We have applied our approach on vulnerabilities of four different OSs (Windows (Microsoft), Mac (Apple), IOS (Cisco) and Linux) and four web browsers [Internet Explorer (Microsoft), Safari (Apple), Firefox (Mozilla), and Chrome (Google)]. For these OSs, our approach when compared to a power-law model without clustering issued from a family of SRMs, gives more accurate curve fittings and predictions in all cases we analyzed. For these web browsers, in all the cases where the power-law model is statistically adequate for modeling (at least one of the models has a p value greater than 0.05), our approach provides more accurate or more conservative curve fitting and forecasting results.

The rest of the paper is structured as follows. Section 2 presents the related work. Section 3 details the dataset and how we processed the data. Section 4 details the analysis. Section 5 presents the obtained results. Section 6 discusses the results and lists the limitations. Finally, Sect. 7 concludes the paper.

2 Related work

Vulnerabilities are software faults which are exploited as a result of security attacks. Thus, vulnerability discovery models (VDMs) and Software Reliability Models

(SRMs) can be considered similar based on the fault detection processes. Thus, the intensity function can represent the detection rate of vulnerabilities [8]. Research has been conducted to create a link between the fault discovery process and the vulnerability discovery process for modeling purposes [9]. Several studies have proposed new SRMs/ VDMs or applied existing models to estimate software security indicators such as total number of residual vulnerabilities in the system, time to next vulnerability (TTNV), vulnerability detection rate [1–4,7–13].

Rescorla [2,3] proposed a VDM to find the number of undiscovered vulnerabilities. In [4–6], Alhazmi and Malaiya proposed regression models to simulate the vulnerability disclosure rate and predict the number of vulnerabilities that may be present but may not yet have been found. Some studies have tried to increase the accuracy of vulnerability modeling. Joh and Malaiya [14] proposed a new approach for modeling the skewness in vulnerability datasets by modifying common S-shaped models like Weibull and Gamma. These models assume that the software under study has a finite number of vulnerabilities to be discovered [8]. Note that, releasing new patches might be accompanied by introducing new vulnerabilities and dynamically change the number of vulnerabilities. The mentioned models provide better curve fitting results than prediction ones. This is because a model may provide an excellent fit for the available data points, but if the detection trends are not consistent with the model, the model doesn't lead to accurate prediction results [14].

In addition to the vulnerabilities publication dates, software source code has been used for vulnerability assessment in the context of VDMs. Kim et al. [15] introduced a VDM based on shared source code measurements among multi-version software systems. In [16], Ozment and Schechter employed a reliability growth model to analyze the security of the OpenBSD OS by examining its source code and the rate at which new code has been introduced. However, source code has proved not to be an accurate measure in terms of prediction [7].

Clustering is a method of structuring data according to similarities and dissimilarities into natural groupings [17]. Clustering for vulnerabilities has been used for splitting real-world exploited vulnerabilities from those which were exploited during software test (proof-of-concept exploits) [18], as well as for detecting exploited vulnerabilities versus non-exploited ones when there is not enough information about some vulnerabilities [19]. Lee et al. [20] investigated a distributed denial of service (DDoS) attack detection method using cluster analysis. Shahzad et al. [21] conducted a descriptive statistical analysis of a large software vulnerability dataset employing clustering on type-based vulnerability data. Huang et al. [22] classified vulnerabilities employing several clustering algorithms to create a relatively objective classification criterion among the vulnerabilities.

3 Dataset and data processing

The data used in this paper has been collected from the National Vulnerability Database (NVD) maintained by NIST. We developed Python scripts to scrape the publicly available data on vulnerabilities from NVD. We then stored the data in our own database, and identified each vulnerability by its Common Vulnerability Enumeration (CVE)

Table 1 Number of vulnerabilities per OS

OS	Windows	Mac	IOS	Linux
# Vulnerabilities	1015	1129	389	1705
# Labeled vulnerabilities	888 (87.5%)	920 (81.5%)	360 (92.5%)	1439 (84.5%)
# Non-labeled vulnerabilities	127 (12.5%)	209 (18.5%)	29 (7.5%)	266 (15.5%)

identifier. We used the CVE ID to compare the reporting date of each vulnerability in NVD, with the dates in other public repositories on vulnerabilities.[1] We then updated the reporting date on our database to the earliest date that a given vulnerability was known in any of these databases. Details of the tool (vepRisk) we developed to gather the data is given in [23]. For the rest of the paper, we will focus on four OSs (Windows, Mac, IOS, and Linux) and four web browsers (Internet Explorer, Safari, Firefox, and Chrome).

3.1 Operating systems

We will focus on the vulnerabilities reported for four well-known OSs: Windows (1995–2016), Mac (1997–2016), IOS (the OS associated with Cisco) (1992–2016), and Linux (1994–2016). We chose these OSs as they are most widely used, and had the most vulnerabilities in NVD. For each OS, we included all the vulnerabilities reported for any of its versions. For instance, all the vulnerabilities reported for mac-os, mac-os-server, mac-os-x, and mac-os-x-server were put together to create a dataset for Mac. We did this to have enough data for each OS. The total number of vulnerabilities in NVD for these OSs is 4238. We used text information within vulnerabilities reports to then label the vulnerabilities. The keywords for labeling (e.g., denial, injection, buffer, execute) were extracted from these reports. Table 1 shows the total number of vulnerabilities as well as the number of labeled and non-labeled (vulnerabilities without any associated text information in the database) vulnerabilities for these OSs.

For the labeled vulnerabilities, we indicate the number and proportion of vulnerabilities associated with a specific keyword. Note that vulnerabilities can be labeled with more than one keyword. Details for OSs are in Table 2.

We treated each keyword as a binary variable. Thus, each vulnerability is represented as a matrix with a binary vector; if a keyword is present in the textual information of a given vulnerability, the value of that keyword is "1" otherwise, the values is "0". We excluded non-labeled vulnerabilities from our clusters. For cluster analysis, we need to ensure that the features (keywords) are not correlated. Therefore, we checked the Pearson correlation coefficient for every two keywords per OS. When we found statistically significant correlation, we merged the correlated keywords with a title which included both terms. For instance, due to the high correlation of 0.99 (p value < 0.001, H_0 :p = 0) between "Execute" and "Code" for all OSs, these terms were treated as "Execute Code". The same applied for the keywords "SQL" and "Injection". No other significant correlation was observed.

[1] cvedetails.com, cxsecurity.com, security-database.com and securityfocus.com.

Table 2 Number of vulnerabilities per type and OS

Keywords	Windows	Mac	IOS	Linux
Denial of service	242 (27.25%)	412 (44.8%)	296 (82.2%)	860 (59.8%)
Execute code	289 (32.5%)	458 (49.8%)	24 (6.7%)	161 (11.2%)
Overflow	174 (19.6%)	338 (36.7%)	29 (8.1%)	298 (20.7%)
SQL injection	0	0	0	4 (0.3%)
Obtain information	49 (5.5%)	113 (12.3%)	9 (2.5%)	229 (15.9%)
Gain privileges	325 (36.6%)	116 (12.6%)	4 (1.1%)	205 (14.25%)
Bypass restriction or similar	62 (7.0%)	115 (12.5%)	38 (10.6%)	112 (7.8%)
Directory traversal	2 (0.2%)	12 (1.3%)	4 (1.1%)	8 (0.6%)
Cross site scripting	10 (1.1%)	15 (1.6%)	2 (0.6%)	11 (0.8%)
Http response splitting	0	2 (0.2%)	0	1 (0.07%)
CSRF	0	2 (0.2%)	1 (0.3%)	0
Memory corruption	59 (6.6%)	145 (15.8%)	5 (1.4%)	70 (4.9%)

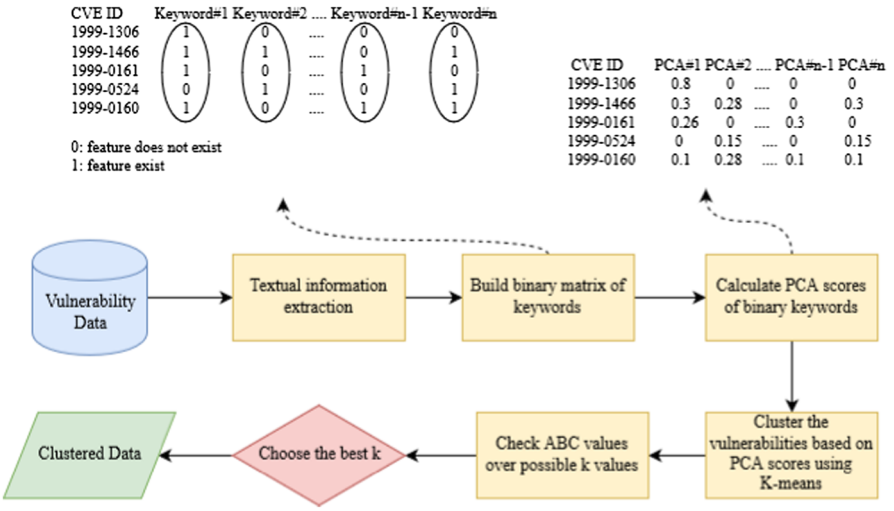

Fig. 1 Diagram of the presented clustering approach

Figure 1 shows the diagram of our clustering approach. For clustering, we used the well-known k-means algorithm for clustering nominal input variables. Since we cannot use binary variables to compute distances within seeds, we used the associated principal component analysis (PCA) scores derived from the linear combinations of binary attributes for each OS. PCA reduces the number of features that might be correlated to independent linear combinations of them [17]. An explanation for using PCA with binary data is provided in [24]. Using PCA scores, the k-means procedure uses the least square method to compute cluster centroids. Each iteration reduces the criterion (e.g., the least squared criterion for Euclidean distance) until convergence is achieved or the maximum iteration number is reached [25].

Table 3 Number of vulnerabilities per type, cluster (Windows)

Keywords	Windows					
	1	2	3	4	5	6
Denial of service	16 (15.5%)	22 (91.7%)	6 (33.3%)	188 (69.1%)	5 (1.7%)	5 (2.9%)
Execute code	98 (95.1%)	5 (20.8%)	15 (83.3%)	0	0	171 (100%)
Overflow	103 (100%)	21 (87.5%)	0	17 (6.25%)	33 (11.0%)	0
SQL injection	0	0	0	0	0	0
Obtain information	1 (1.0%)	0	1 (5.55%)	43 (15.8%)	4 (1.33%)	0
Gain privileges	2 (1.9%)	14 (58.3%)	0	0	300 (100%)	9 (5.3%)
Bypass restriction or similar	0	0	0	56 (20.6%)	5 (1.7%)	1 (0.6%)
Directory traversal	0	0	0	1 (0.4%)	1 (0.3%)	0
Cross site scripting	0	0	0	8 (2.9%)	0	2 (1.2%)
Http response splitting	0	0	0	0	0	0
CSRF	0	0	0	0	0	0
Memory corruption	9 (8.7%)	23 (95.8%)	18 (100%)	0	9 (3.0%)	0
# Vulnerabilities	103	24	18	272	300	171

Since we are using an unsupervised clustering approach, after clustering data for different values of k, we need to use a metric to choose the number of clusters (the best k value). To find the number of clusters, we used the aligned box criterion (ABC) method. The cubic clustering criterion (CCC) is another common metric used to find the most suitable number of clusters [26]. Tibshirani et al. [27] proposed a gap statistics method which leverages Monte Carlo simulation for finding the number of clusters in a database. However, it has been shown that ABC improves the CCC and gap statistics methods by leveraging a high-performance machine-learning based analysis structure [25]. In addition, we used a within-cluster dispersion as an error measure (also called a "Gap") by the ABC method [27]. In order to find the number of clusters, we applied the ABC method that compares the calculated Gap values over a range of possible k values. The appropriate number of clusters occurs at the maximum peak value in Gap (k) [25]. We obtained 6, 6, 7, 7 clusters for Windows, Mac, IOS, and Linux, respectively. Tables 3, 4, 5 and 6 list the keywords associated with each of the six/seven clusters for Windows, Mac, IOS, and Linux respectively.

Table 4 Number of vulnerabilities per type, cluster (Mac)

Keywords	Mac					
	1	2	3	4	5	6
Denial of service	174 (64.2%)	0	120 (39.0%)	0	0	118 (100%)
Execute code	253 (93.4%)	0	0	3 (2.6%)	90 (96.8%)	112 (94.9%)
Overflow	101 (37.3%)	0	23 (7.5%)	3 (2.6%)	93 (100%)	118 (100%)
SQL injection	0	0	0	0	0	0
Obtain information	1 (0.4%)	1 (6.7%)	103 (33.4%)	8 (7.0%)	0	0
Gain privileges	12 (4.4%)	0	99 (32.1%)	4 (3.5%)	0	1 (0.8%)
Bypass restriction or similar	0	0	0	115 (100%)	0	0
Directory traversal	3 (1.1%)	0	8 (2.6%)	1 (0.9%)	0	0
Cross site scripting	0	15 (100%)	0	0	0	0
Http response splitting	0	2 (13.3%)	0	0	0	0
CSRF	0	0	1 (0.3%)	1 (0.9%)	0	0
Memory corruption	139 (51.3%)	0	5 (1.6%)	1 (0.9%)	0	0
# Vulnerabilities	271	15	308	115	93	118

We assumed that the keywords which covered at least 60% of vulnerabilities in each cluster can be good representatives of relative clusters. Since none of the keywords reached the weight threshold of 0.6 in the third cluster associated with Mac, the keyword with the greatest weight (Denial of Service) was selected as the cluster's label. There is only one common cluster name among all the OSs. There are also clusters with similar names within some OSs.

However, analyzing their linear correlation, we did not find any significant relationship based on the Pearson correlation test. Table 7 shows the cluster summaries for the OSs.

3.2 Web browsers

We will focus on the vulnerabilities reported for four well-known web browsers: Internet Explorer (1997–2016), Safari (2003–2016), Firefox (2003–2016), and Chrome

Table 5 Number of vulnerabilities per type, cluster (IOS)

Keywords	IOS						
	1	2	3	4	5	6	7
Denial of service	0	8 (100%)	19 (73.1%)	3 (100%)	4 (30.8%)	0	262 (100%)
Execute code	3 (6.8%)	0	8 (30.8%)	0	13 (100%)	0	0
Overflow	1 (2.3%)	2 (25%)	26 (100%)	0	0	0	0
SQL injection	0	0	0	0	0	0	0
Obtain information	6 (13.6%)	3 (37.5%)	0	0	0	0	0
Gain privileges	1 (2.3%)	0	0	0	3 (23.1%)	0	0
Bypass restriction or similar	35 (79.5%)	0	0	3 (100%)	0	0	0
Directory traversal	0	0	0	0	0	4 (100%)	0
Cross site scripting	2 (4.5%)	0	0	0	0	0	0
Http response splitting	0	0	0	0	0	0	0
CSRF	0	0	0	0	1 (7.7%)	0	0
Memory corruption	0	5 (62.5%)	0	0	0	0	0
# Vulnerabilities	44	8	26	3	13	4	262

(2008–2016). These browsers were selected since they are widely used and had the highest number of vulnerabilities in the database. Similar to what we did for OSs, we considered, for each browser, all the vulnerabilities reported for any of its versions. As an example, all the vulnerabilities reported for ie, ieexplorer, and ie-for-macintosh were combined under Internet Explorer. Thus, we had enough data for each browser. The total number of vulnerabilities for these browsers is 2434. The process of extracting text information from vulnerability reports and labeling vulnerabilities is the same as for the OSs. The total number of vulnerabilities as well as the number of labeled and non-labeled vulnerabilities for the selected browsers are presented in Table 8. Table 9 shows the number and proportion of vulnerabilities associated with a specific keyword for the labeled vulnerabilities. As previously mentioned, vulnerabilities can be labeled with more than one keyword.

Following the same clustering approach we explained for the OSs, we obtained the following number of clusters: Internet Explorer (5), Safari (3), Firefox (5), and Chrome (5). We used the same threshold value we had selected before for OSs since we believe

Table 6 Number of vulnerabilities per type, cluster (Linux)

Keywords	Linux						
	1	2	3	4	5	6	7
Denial of service	136 (53.1%)	53 (100%)	70 (100%)	600 (100%)	1 (0.9%)	0	0
Execute code	126 (49.2%)	0	7 (10.0%)	17 (2.8%)	8 (7.1%)	2 (1.4%)	1 (0.5%)
Overflow	208 (81.3%)	15 (28.3%)	36 (51.4%)	0	0	31 (21.2%)	8 (4.0%
SQL injection	2 (0.7%)	0	0	0	2 (1.8%)	0	0
Obtain information	4 (1.6%)	2 (3.8%)	3 (4.3%)	10 (1.7%)	3 (2.7%)	5 (3.4%)	202 (100%)
Gain privileges	0	53 (100%)	15 (21.4%)	0	3 (2.7%)	134 (91.8%)	0
Bypass restriction or similar	1 (0.4%)	0	0	5 (0.8%)	106 (94.6%)	0	0
Directory traversal	0	0	0	0	7 (6.25%)	1 (0.7%)	0
Cross site scripting	0	0	0	0	0	11 (7.5%)	0
Http response splitting	0	0	0	0	0	1	0
CSRF	0	0	0	0	0	0	0
Memory corruption	0	0	70 (100%)	0	0	0	0
# Vulnerabilities	256	53	70	600	112	146	202

that the keywords which covered at least 60% of vulnerabilities in each cluster can be good representatives of relative clusters. For those clusters, for which none of the keywords reach the weight threshold (i.e., cluster #2 in Safari), the keyword with the greatest weight becomes the cluster's label. The cluster summaries for the browsers are shown in Table 10. More details about the clusters and frequency of the keywords associated with the clusters are provided in the following report online[2] (we had to remove them from the paper due to space constraints).

4 Analysis

A nonhomogeneous Poisson process (NHPP) is often used when modeling the mean cumulative number of failures (MCF) for repairable systems and for software reliability evaluations. The core assumption is that the number of detected failures follows a

[2] https://figshare.com/s/85366e113d19e4baeae3.

Table 7 Cluster composition for operating systems

OS	Cluster number	Prevalent keywords	Cluster name
Windows	1	Execute code, Overflow	EO
	2	DoS, Overflow, Memory corruption	DOM
	3	Execute code, Memory corruption	EM
	4	DoS	D
	5	Gain privileges	G
	6	Execute code	E
Mac	1	DoS, Execute code	DE
	2	Cross site scripting	C
	3	DoS	D
	4	Bypass a restriction	B
	5	Execute code, Overflow	EO
	6	DoS, Execute code, Overflow	DEO
IOS	1	Bypass a restriction	B
	2	DoS, Memory corruption	DM
	3	DoS, Overflow	DO
	4	DoS, Bypass a restriction	DB
	5	Execute code	E
	6	Directory traversal	DT
	7	DoS	D
Linux	1	Overflow	O
	2	DoS, Gain privileges	DG
	3	DoS, Memory corruption	DM
	4	DoS	D
	5	Bypass a restriction	B
	6	Gain privileges	G
	7	Obtain information	O

Table 8 Number of vulnerabilities per web browser

Web browser	Explorer	Safari	Firefox	Chrome
# Vulnerabilities	379	248	890	917
# Labeled vulnerabilities	248 (65.4%)	210 (84.7%)	720 (80.8%)	796 (86.8%)
# Non-labeled vulnerabilities	131 (34.6%)	38 (15.3%)	170 (19.2%)	121 (13.2%)

nonhomogeneous Poisson process. In the case of NHPP-based repairable systems, the intensity function $\lambda(t) = dE[\Lambda(t)]/dt$ is often assumed to be a monotonic function of t. Similarly, in NHPP-based software reliability models (SRMs), the intensity function (the detection rate of software errors) is considered to be a monotonic function [28].

Let us expand the discussion for a software when there exists more than one type of error. When any type of error independently causes the software normal function

Table 9 Number of vulnerabilities per type and web browser

Keywords	Explorer	Safari	Firefox	Chrome
Denial of service	90 (36.3%)	162 (77.1%)	354 (49.2%)	634 (79.65%)
Execute code	114 (46.0%)	150 (71.4%)	432 (60.0%)	53 (6.7%)
Overflow	41 (16.5%)	98 (46.7%)	158 (21.9%)	203 (25.5%)
SQL injection	0	0	0	0
Obtain information	18 (7.3%)	17 (8.1%)	63 (8.75%)	36 (4.5%)
Gain privileges	0	0	15 (2.1%)	0
Bypass restriction or similar	32 (12.9%)	19 (9.05%)	95 (13.2%)	80 (10.05%)
Directory traversal	5 (2.0%)	2 (0.95%)	6 (0.8%)	2 (0.25%)
Cross site scripting	23 (9.3%)	10 (4.8%)	75 (10.4%)	29 (3.6%)
Http response splitting	1 (0.4%)	0	1 (0.1%)	0
CSRF	0	0	7 (1.0%)	3 (0.4%)
Memory corruption	33 (13.3%)	130 (61.9%)	220 (30.6%)	59 (7.4%)

Table 10 Cluster composition for web browsers

Web browser	Cluster number	Prevalent keywords	Cluster name
Internet Explorer	1	DoS	D
	2	Execute code, Memory corruption, DoS	EMD
	3	Bypass a restriction	B
	4	Execute code	E
	5	Cross Site Scripting	C
Safari	1	Execute code, Memory corruption, DoS	EMD
	2	DoS (0.29), Bypass a restriction (0.29)	DB
	3	Execute code, Memory corruption, DoS, Overflow	EMDO
Firefox	1	Execute code, Overflow	EO
	2	Execute code, Memory corruption, DoS	EMD
	3	Bypass a restriction (0.34)	B
	4	Execute code, Memory corruption, DoS, Overflow	EMDO
	5	Execute code	E
Chrome	1	Bypass a restriction	B
	2	Memory corruption, DoS, Overflow	MDO
	3	Memory corruption, DoS	MD
	4	DoS, Overflow	DO
	5	DoS	D

to be compromised, then the superposition model represents the software failures. Let us assume that we are dealing with vulnerabilities classified into independent clusters. Let $\Lambda_j(t)$ denote the NHPP for the vulnerabilities from the jth cluster in (0 t] with intensity function $\lambda_j(t|\alpha_j, \beta_j)$ where the function form of $\lambda_j(t|\alpha_j, \beta_j)$ is given and the values of the parameters α_j, β_j are unknown. It is assumed that the

number of vulnerabilities from any jth cluster $\Lambda_j(t)$, $j = 1, 2 \ldots J$ is independent. A process $\Lambda(t) = \sum_{j=1}^{J} \Lambda_j(t)$, which counts the total number of vulnerabilities in the interval (0 t] for the superposition model, is also a non-homogeneous Poisson process with an intensity function $\lambda(t|\alpha, \beta) = \lambda_1(t|\alpha_1, \beta_1) + \cdots + \lambda_1(t|\alpha_J, \beta_J)$, where $\alpha = \{\alpha_1, \ldots, \alpha_J\}$, $\beta = \{\beta_1, \ldots, \beta_J\}$. Since the superposition model remains an NHPP (all intensity functions are NHPPs), the associated superposition model can be applicable [28]. Two important cases of NHPPs consist of the related intensity function following a power-law and log linear (exponential) function of time [29,30]. In such case, the equations become:

$$\lambda_j(t|\alpha_j, \beta_j) = \frac{\alpha_j}{\beta_j}\left(\frac{t}{\beta_j}\right)^{\alpha_j - 1} = \frac{\alpha_j t^{\alpha_j - 1}}{\beta_j^{\alpha_j}} \tag{1}$$

$$\Lambda(t) = \int_0^t \sum_{j=1}^{J} \lambda_j(t|\alpha_j, \beta_j)dt, \alpha_j > 0, \beta_j > 0 \tag{2}$$

In this paper, we expect to obtain better assessment results when relaxing the monotonicity assumption of the intensity function that is prevalent in SRMs and VDMs. We created independent clusters that can be modeled using separate NHPPs. In this paper, we selected the power-law model since Allodi [31] showed that vulnerabilities may follow a power-law distribution. In addition, the power-law model has been widely used in research on software reliability analysis [32–34]. We considered two models for this paper. The first model is a NHPP-based SRM, which uses non-clustered data (including all the labeled and non-labeled vulnerabilities). The second model is the superposition of the NHPPs fitted to the clustered data (only the labeled vulnerabilities can be used to create the clusters), which relaxes the monotonicity assumption of the intensity function. The purpose of both models is to fit and predict the total number of reported vulnerabilities (labeled and non-labeled).

The analysis was done in two steps. First, we used the time difference between vulnerability report dates to find the model parameters from the process of fitting NHPPs to the data (clustered and non-clustered). The models were fitted to the datasets using a non-linear regression method described in [35] that uses a minimization algorithm called "$Levenberg - Marquardt$" to estimate the parameter values. Non-homogeneity of the clusters were also validated by looking at Laplace-trend test results provided by MiniTab 16 to see whether there were meaningful trends in the clusters. Second, we used the estimated parameters and the models, and simulated corresponding MCFs (one MCF for clustered data, and one for non-clustered data) starting from $t_0 = 0$ and time interval of 30 days.

5 Results

In this section we will provide the results regarding estimation (comparing the results between clustered data and non-clustered data) and forecasting (comparing the obtained predictions with clustered data and non-clustered data using a subset of

the data). We first discuss the results obtained for the four OSs and then for the four web browsers.

5.1 Operating systems

Figure 2 shows the observed vulnerability data, the MCF obtained without clustering the data and the superimposed MCF when clustering is applied for the four OSs. For Windows, the MCF with clustering is more conservative during roughly the first 3000 days then the MCF without clustering becomes more conservative. The real data crosses the estimates between roughly 2000 and 3000 days. Besides this period, the MCF with and without clustering provide more conservative estimates. For IOS, the MCF with and without clustering as well as the real data are almost overlapping. For Mac and Linux, the MCF with clustering is above the MCF without clustering, providing a more conservative estimation. The real data is above the MCF without clustering and for a short period above the MCF with clustering. Thus for Mac and Linux, the MCF with clustering provide more conservative and accurate estimates.

The analysis of forecasting is done for the final third of the time period from the beginning of the vulnerability discovery process. During the training period (first two thirds of the time period), all the available data are used to estimate model parameters. Figure 3 shows the forecast of the number of vulnerabilities based on the MCFs calculated with and without clustering compared to the observed vulnerabilities. For the vulnerabilities associated with Windows, the forecast with clustering leads to more conservative and more accurate estimates compared to non-clustering. In addition, the forecast without clustering remains below the observed vulnerability data (real data) for the prediction time period which started after day 5088. Both models lead to more conservative predictions for the vulnerabilities associated with Mac. However, the clustering-based MCF trajectory provides more accurate predictions. For IOS the forecast without clustering is more conservative compared to the forecast with clustering. When the MCF without clustering remains above the real data (which is good), it is not the case of the forecast with clustering where the real data crosses the forecast after day 7500. For Linux, the predictions with clustering and the real observed data are close but the forecast remains above the real data and thus provides conservative estimation (which is good). The clustering-based MCF is, however, more accurate than that of the non-clustering MCF.

We applied the Chi-square (χ^2) goodness of fit test [14] to see how well each model fits the datasets for both estimation and forecasting. The Chi-square statistic is calculated using the following equation:

$$\chi^2 = \sum_{i=1}^{N} \frac{(S_i - O_i)^2}{O_i} \tag{3}$$

where S_i and O_i are the simulated and expected observed values at ith time point, respectively. N is the number of observations (the time blocks used for simulation). For the fit to be acceptable, the corresponding χ^2 critical value should be greater than the χ^2 statistic value for the given alpha level and degrees of freedom. We selected an

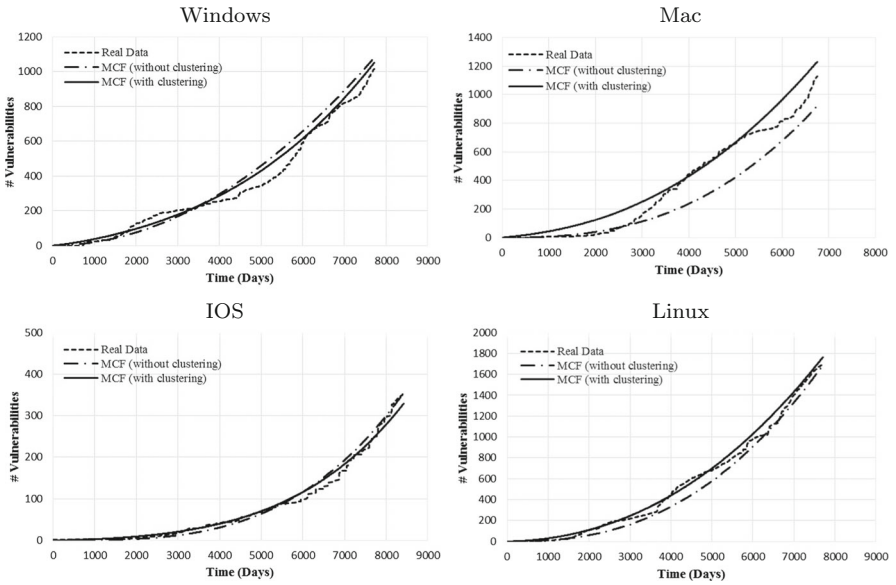

Fig. 2 Comparison of clustered and non-clustered MCFs with vulnerability data

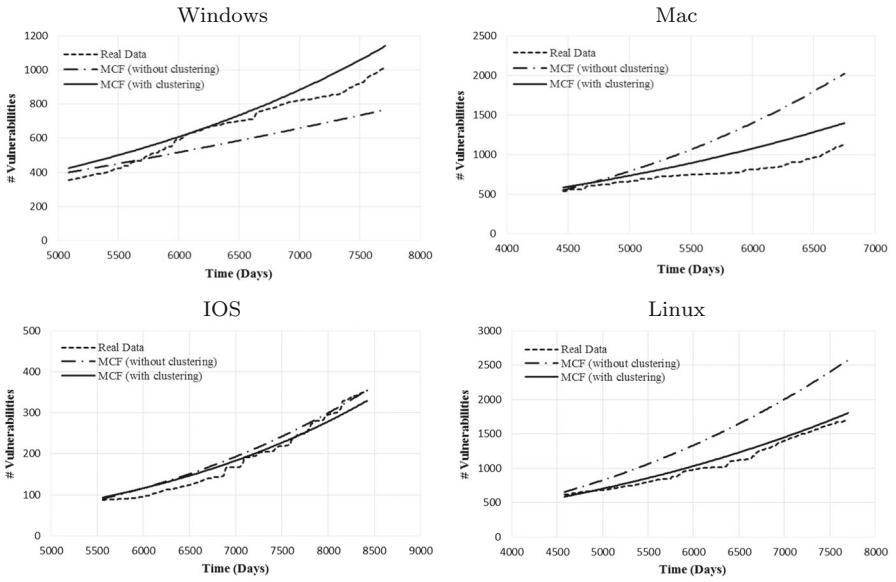

Fig. 3 Comparison of clustered and non-clustered forecasts with vulnerability data

alpha level of 0.05. The null hypothesis indicates that the actual distribution is well described by the fitted model. Hence, if the p value of the χ^2 test is below 0.05, then the fit will be considered to be unsatisfactory. A p value closer to 1 indicates a better fit.

R^2 is another fitting statistic widely used in regression analysis [17,36]. R^2 values close to 1 indicates a good fit. We also considered an additional error indicator to compare the accuracy of the results derived from both models. The normalized root mean square error (NRMSE) is often used. However, Mentaschi et al. [37] showed that for some applications (e.g., high fluctuation of real data) the higher values of NRMSE are not always a reliable indicator of the accuracy of simulations. To remedy the situation, a corrected estimator HH was proposed by Hanna and Heinold [38]:

$$HH = \sqrt{\frac{\sum_{i=1}^{N}(S_i - O_i)^2}{\sum_{i=1}^{N} S_i O_i}} \tag{4}$$

where S_i is the ith simulated data, O_i is the ith observation and N is the number of observations (the time blocks used for simulation). The closer to zero HH is, the more accurate the model.

Table 11 contains the Chi-square goodness of fit test for the clustering-based MCF and the MCF without clustering, the values of R^2, and HH for the vulnerabilities of the four OSs in our study. We considered the entire dataset for analyzing estimation accuracy. When considering the entire dataset, both estimations (with/without clustering) are statistically sound for all OSs but one (Mac, without clustering) with p values greater than 0.05. The Chi-square test results indicate that both fits are reasonably good in most cases except the case associated with the non-clustered based MCF on Mac data. The R^2 statistics show that estimations based on clustering are more accurate than the ones without clustering. HH results also show that clustering based estimations came up with smaller errors compared with non-clustering. For the four OSs, the estimations based on clustering were more accurate in all cases. In addition, the MCF model without clustering was not statistically adequate to model the vulnerability data in one case (Mac). Table 12 contains the Chi-square goodness of fit test for prediction values of the clustering and non-clustering-based MCF, the R^2 and HH prediction values for the vulnerabilities of the four OSs. We considered the common 66% splits between training and forecasting which means all the available data in the first two thirds of the study time period were used to estimate model parameters.

For the vulnerabilities associated with Windows, Mac, and Linux, all considered training/forecasting results using non-clustered data lead to statistically inadequate fits since all the relative p values are zeros. For vulnerabilities associated with IOS, while both models came up with adequate fits based upon the Chi-squared test result, the clustering-based forecast is more accurate due to higher R^2 and lower HH values. For the other OSs, the predictions associated with clustered data are statistically sound with p values greater than 0.05 and reasonably good R^2 and HH values. Thus, the forecasts based on clustering were more accurate in all cases. In addition, the MCF model without clustering was not statistically adequate to model the vulnerability data in three cases.

Table 11 Estimation accuracy for the four operating systems

Estimation	With clustering			Without clustering		
	p value	R-sq	HH	*p* value	R-sq	HH
Windows	1	0.975	0.098	1	0.945	0.145
Mac	0.052	0.851	0.242	0.028	0.837	0.326
IOS	1	0.987	0.084	1	0.972	0.098
Linux	1	0.995	0.06	1	0.992	0.099

Table 12 Forecasting accuracy for the four operating systems

Forecasting	With clustering			Without clustering		
	p value	R-sq	HH	*p* value	R-sq	HH
Windows	0.792	0.838	0.104	0	0.558	0.196
Mac	0.115	0.577	0.238	0	− 0.316	0.514
IOS	0.992	0.902	0.122	0.977	0.896	0.135
Linux	0.708	0.833	0.102	0	− 1.822	0.367

5.2 Web browsers

Figure 4 shows the observed vulnerability data, the MCF obtained without clustering the data and the superimposed MCF when clustering is applied for the four web browsers. For Internet Explorer, the observed number of vulnerabilities almost leveled off after 4000 days which means the relative vulnerability detection rate decreased. Both MCFs with and without clustering seem inadequate in modeling Internet Explorer data since they are power-law based equations and are designed to model data with constant or increasing discovery rate. Thus, both MCFs show high deviations from the real data. The number of vulnerabilities associated with Chrome also experiences a similar condition after day 1500, and both MCFs seem inappropriate for modeling such S-shaped data. However, in the first 1500 days the MCF with clustering is more conservative than the MCF without clustering. For Safari, despite large fluctuation at day 3400, both MCFs are capable of providing good fits to the real data. However, the MCF with clustering seems to be closer to the observed data, most of the time, but is less conservative. For Firefox, which has almost constant discovery rate, the MCF with clustering is more accurate and more conservative than the MCF without clustering, especially after day 1500. In summary, the MCF with clustering provides more accurate estimations for Safari and Firefox. However, for Internet Explorer and Chrome, the power-law model is not adequate for curve-fitting purposes.

Like for the OSs, the analysis of forecasting is done for the final third of the time period. During the training period (first two thirds of the time period), the data is used to estimate model parameters. Figure 5 shows the forecast of the number of vulnerabilities based on the MCFs calculated with and without clustering compared to the web browsers' observed number of vulnerabilities.

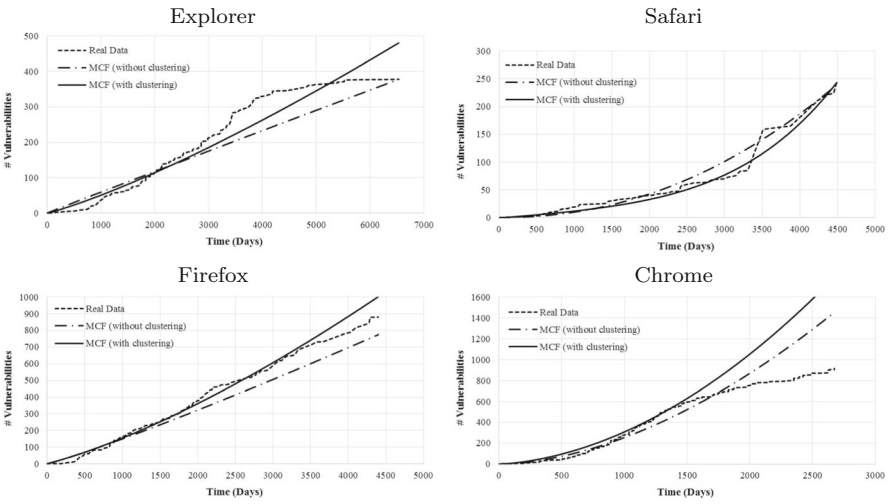

Fig. 4 Comparison of clustered and non-clustered MCFs with vulnerability data

For the vulnerabilities associated with Internet Explorer and Chrome, similar to what we observed for estimation, both MCFs forecasting results are distant from the real data. Since both MCFs (with/without clustering) for Internet Explorer and Chrome highly deviate from the real data, we observe that the power-law model is not suitable for modeling this data. For Safari, the forecast with clustering provides more accurate predictions compared to the forecast without clustering. The MCF without clustering was unable to provide accurate forecasts after day 3300. For Firefox, both MCFs provide very accurate forecasts until day 3530. Starting from day 3530, both MCFs provide conservative but less accurate forecasts, (the MCF with clustering provides slightly conservative results).

Table 13 contains the Chi-square goodness of fit test for the clustering-based MCF and the MCF without clustering, the values of R^2, and HH for the vulnerabilities of the four web browsers. When analyzing estimation accuracy, both MCFs (with/without clustering) are statistically significant for two browsers (Safari and Firefox) with p values greater than 0.05. The Chi-square test results indicate that both MCFs are not statistically significant for Internet Explorer and Chrome which means that the selected power-law model is not suitable for modeling the data with such attributes (i.e., data with decreasing discovery rate in their overall discovery trend). The R^2 statistics/values for Safari and Firefox are almost equal per case, and indicate that both models have provided good fit to the real data. The HH results show that clustering based estimations have smaller errors compared with non-clustering for Safari and Firefox.

Table 14 contains the Chi-square goodness of fit test for prediction values of the clustering and non-clustering-based MCF, the R^2 and HH prediction values for the vulnerabilities of the four web browsers in our study. We considered the common 66% splits between training and forecasting which means all the available data in the first two thirds of the study time period were used to estimate model parameters.

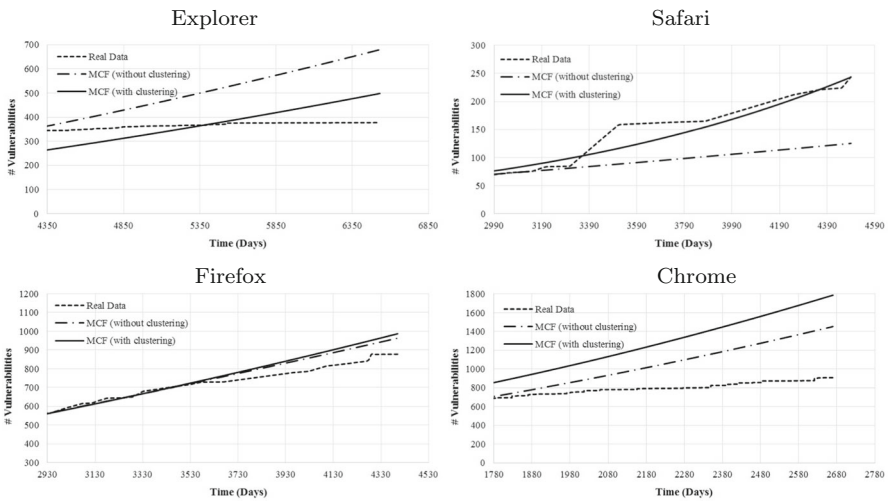

Fig. 5 Comparison of clustered and non-clustered forecasts with vulnerability data

Table 13 Estimation accuracy for the four Web Browsers

Estimation	With clustering			Without clustering		
	p value	R-sq	HH	*p* value	R-sq	HH
Explorer	0	0.92	0.31	0	0.95	0.21
Safari	0.96	0.98	0.12	0.90	0.97	0.13
Firefox	1	0.99	0.084	0.98	0.99	0.15
Chrome	0	0.88	0.455	0	0.88	0.295

Table 14 Forecasting accuracy for the four Web Browsers

Forecasting	With clustering			Without clustering		
	p value	R-sq	HH	*p* value	R-sq	HH
Explorer	0	0.89	0.16	0	0.89	0.39
Safari	1	0.94	0.11	0	0.90	0.54
Firefox	0.98	0.98	0.071	0.98	0.98	0.057
Chrome	0	0.94	0.51	0	0.94	0.315

For the vulnerabilities associated with Internet Explorer and Chrome, all considered forecasting results using both clustered/non-clustered data lead to p values less than 0.5. However, it will not be a problem since due to high fluctuation of vulnerability data, HH values are recommended to compare the modeling strategies in terms of prediction. Almost all the research in this area used normalized predictability metrics like HH, NRMSE, and average error (AE) for making predictability comparisons between SRMs/VDMs.

For Safari, the forecast results without clustering are also statistically insignificant, while the MCF with clustering leads to statistically significant results. The HH value for

the MCF with clustering is also smaller than that without clustering. For Firefox, both models lead to statistically significant results. The forecasts are very close based on the R-squared values. However, the HH values show that the model without clustering provides more accurate results than the model with clustering.

6 Limitations

The main limitation of the work we present in the paper is with regard to using SRMs as VDMs. Software reliability models usually assume that the time between failures represents total usage time of that product. What we are using is calendar time, which may not be a good proxy for usage. Crucially the difference in security is the difficulty in estimating the "attacker effort"—the total amount of time that an attacker spends in finding a vulnerability—which is something that is not needed for reliability (we assume the users accidentally encounter faults that lead to failures, hence usage time is a good enough proxy for time between failures). A useful discussion of this is given in [39]. Note that this limitation is not only for our work, but applies to research that uses SRMs as VDM and utilizes vulnerability data. However, attacker effort is something that is very difficult to estimate and quantify. The purpose of our research is hence to make as good a use as possible of the publicly available security data to help with decision making. But at the same time to be clear about the limitations on what we can conclude from this analysis. The best we can say from the analysis we present is "the total number of vulnerabilities that will be reported in the NVD over an interval t for product x is y with confidence z". And we show that we can do this prediction better with clustering than without clustering for four of the largest and most commonly used operating system and web browsers families. For some decision makers this may be a valuable piece of additional information, which they can use in conjunction with data they have from their own installations, when deciding on security operating system/web browser/product choices, and provisioning of security support services to deal with new vulnerabilities.

We have only applied the approach to four well-known operating systems and web browsers that had the largest number of vulnerabilities compared to others. We do not know yet how well this works for other operating systems and web browsers or other applications like databases, though we plan to extend this work in the future.

7 Conclusions

We presented an approach that: first, uses existing clustering techniques to group vulnerabilities into distinct clusters; second, uses an existing nonhomogeneous Poisson process (NHPP) Software Reliability Model (SRM) to make predictions on the number of new vulnerabilities that will be discovered in a given time period for each of those clusters for a given OS/web browser; and finally, superimpose the SRMs used for each cluster together into a single model for predicting the number of new vulnerabilities that will be discovered in a given time period for a given OS/web browser.

We provided results from applying our approach to vulnerabilities of four different OSs (Windows, Mac, IOS, and Linux) and four web browsers (Internet Explorer, Safari, Firefox, and Chrome), and comparisons of the predictive accuracy of our approach compared with an NHPP model (with monotonic intensity function) that does not use clustering. For the OSs, we found that our approach with clustering, compared with the same modeling mechanism without clustering:

- Is statistically adequate in terms of model fitting and forecasting based upon the Chi-squared goodness of fit test results for all cases we analyzed, while the model without clustering was not statistically sound in 4 out of 8 cases analyzed
- Gives more conservative forecasting results in all cases while the model without clustering was not conservative for Windows
- Gives more accurate results for all the cases analyzed compared to non-clustering.

For the web browsers, we found that our approach with clustering, compared with the same modeling mechanism without clustering:

- Gives more accurate results in terms of curve-fitting for all cases where the power-law model was statistically sound to model the data (i.e., Safari and Firefox)
- Leads to statistically insignificant results in terms of prediction for two browsers with or without clustering . For Firefox, the forecasting results are less accurate but slightly more conservative for non-clustering compared to clustering.

These results look encouraging, especially as they have been applied for some of the most widely used software products (operating systems and web browsers) which tend to have the highest number of vulnerabilities reported.

Acknowledgements This research is supported by NSF Award #1223634, and the UK EPSRC project D3S (Diversity and defence in depth for security: a probabilistic approach) and the European Commission through the H2020 programme under Grant Agreement 700692 (DiSIEM).

References

1. Lyu MR (ed) (1996) Handbook of software reliability engineering. IEEE Computer Society Press, Los Alamitos; McGraw Hill, New York
2. Rescorla E (2003) Security holes... Who cares? In: Proceedings of the 12th conference on USENIX Security Symposium, pp 75–90
3. Rescorla E (2005) Is finding security holes a good idea? IEEE Secur Priv Mag 3:14–19
4. Alhazmi O, Malaiya Y (2005) Quantitative vulnerability assessment of systems software. IEEE, pp 615–620
5. Woo S, Alhazmi O, Malaiya Y (2006) Assessing Vulnerabilities in Apache and IIS HTTP Servers. IEEE, pp 103–110
6. Alhazmi O, Malaiya Y (2008) Application of vulnerability discovery models to major operating systems. IEEE Trans Reliab 57:14–22
7. Ozment JA (2007) Vulnerability discovery & software security. Ph.D. thesis, University of Cambridge
8. Okamura H, Tokuzane M, Dohi T (2009) Optimal security patch release timing under non-homogeneous vulnerability-discovery processes. In: Proceedings of the 20th international symposium on software reliability engineering. IEEE, pp 120–128

9. Verissimo P, Neves N, Cachin C, Poritz J, Powell D, Deswarte Y, Stroud R, Welch I (2006) Intrusion-tolerant middleware: the road to automatic security. IEEE Secur Priv Mag 4:54–62
10. Okamura H, Tokuzane M, Dohi T (2013) Quantitative security evaluation for software system from vulnerability database. J Softw Eng Appl 06:15
11. Arbaugh WA, Fithen WL, McHugh J (2000) Windows of vulnerability: a case study analysis. Computer 33:52–59
12. Frei S, May M, Fiedler U, Plattner B (2006) Large-scale vulnerability analysis. In: Proceedings of the 2006 SIGCOMM workshop on large-scale attack defense, LSAD '06, New York, NY, USA. ACM, pp 131–138
13. Frei S, Schatzmann D, Plattner B, Trammell B (2010) Modeling the security ecosystem—the dynamics of (in)security. In: Pym DJ, Ioannidis C (eds) Economics of information security and privacy. Springer, Boston, pp 79–106
14. Joh H, Malaiya YK (2014) Modeling skewness in vulnerability discovery: modeling skewness in vulnerability discovery. Qual Reliab Eng Int 30:1445–1459
15. Kim J, Malaiya YK, Ray I (2007) Vulnerability discovery in multi-version software systems. In: 10th IEEE high assurance systems engineering symposium. HASE '07, pp 141–148
16. Ozment A, Schechter SE (2006) Milk or wine: does software security improve with age? In: Proceedings of the 15th USENIX Security Symposium. USENIX Association, Berkeley, CA, USA
17. Johnson RA, Wichern DW (2007) Applied multivariate statistical analysis, 6th edn. Pearson Prentice Hall, Upper Saddle River, OCLC: ocm70867129
18. Sabottke C, Suciu O, Dumitras T (2015) Vulnerability disclosure in the age of social media: exploiting twitter for predicting real-world exploits. In: USENIX security symposium, pp 1041–1056
19. Younis A, Malaiya YK, Ray I (2016) Assessing vulnerability exploitability risk using software properties. Softw Qual J 24:159–202
20. Lee K, Kim J, Kwon KH, Han Y, Kim S (2008) DDoS attack detection method using cluster analysis. Expert Syst Appl 34:1659–1665
21. Shahzad M, Shafiq MZ, Liu AX (2012) A large scale exploratory analysis of software vulnerability life cycles. In: Proceedings of the 34th international conference on software engineering, ICSE '12, Piscataway, NJ, USA. IEEE Press, pp 771–781
22. Huang S, Tang H, Zhang M, Tian J (2010) Text clustering on national vulnerability database. In: 2010 Second international conference on computer engineering and applications, vol 2, pp 295–299
23. Andongabo A, Gashi I (2017) vepRisk—a web based analysis tool for public security data. IEEE, pp 135–138
24. Gower JC (1966) Some distance properties of latent root and vector methods used in multivariate analysis. Biometrika 53:325–338
25. SAS Institute Inc (2016) SAS® Enterprise Miner™ 14.2: high-performance procedures. SAS Institute Inc., Cary, NC
26. Sarle WS (1983) Cubic clustering criterion. SAS Technical Report A-108. SAS Institution Inc., Cary, NC
27. Tibshirani R, Walther G, Hastie T (2001) Estimating the number of clusters in a data set via the gap statistic. J R Stat Soc Ser B (Stat Methodol) 63:411–423
28. Yang TY, Kuo L (1999) Bayesian computation for the superposition of nonhomogeneous poisson processes. Can J Stat 27:547–556
29. Modarres M, Kaminskiy MP, Krivtsov V (2016) Reliability engineering and risk analysis: a practical guide. CRC Press, Boca Raton
30. Zhao Z, Zhang Y, Liu G, Qiu J (2017) Statistical analysis of time-varying characteristics of testability index based on NHPP. IEEE Access 5:4759–4768
31. Allodi L (2015) The heavy tails of vulnerability exploitation. In: Engineering secure software and systems. Lecture notes in computer science. Springer, Cham, pp 133–148
32. Yoo T-H, Lee J-K, Seo Y-J (2016) A relative research of the software NHPP reliability based on weibull extension distribution and power law model. Indian J Sci Technol 9(46). https://doi.org/10.17485/ijst/2016/v9i46/107199
33. Tae-Hyun Y (2015) The infinite NHPP software reliability model based on monotonic intensity function. Indian J Sci Technol 8(14). https://doi.org/10.17485/ijst/2015/v8i14/68342
34. Kim H-C (2015) The assessing comparative study for statistical process control of software reliability model based on Musa-Okumo and power-law type. J Korea Inst Inf Electron Commun Technol 8(6):483–490

35. Nguyen VH, Dashevskyi S, Massacci F (2016) An automatic method for assessing the versions affected by a vulnerability. Empir Softw Eng 21:2268–2297
36. Gujarati DN, Porter DC (2009) Basic econometrics. McGraw-Hill Irwin. Google-Books-ID: 6l1CPgAACAAJ
37. Mentaschi L, Besio G, Cassola F, Mazzino A (2013) Problems in RMSE-based wave model validations. Ocean Model 72:53–58
38. Hanna SR, Heinold DW, A. P. I. H. a. E. A. Dept, E. R. T. Inc (1985) Development and application of a simple method for evaluating air quality models. American Petroleum Institute. Google-Books-ID: lKrpAAAAMAAJ
39. Littlewood B, Brocklehurst S, Fenton N, Mellor P, Page S, Wright D, Dobson J, McDermid J, Gollmann D (1993) Towards operational measures of computer security. J Comput Secur 2:211–229